FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

Thesis of MSc in Mathematics

Unsupervised bilingual dictionary induction with stable GANs

Author: Xuwen Zhang <dlv618@alumni.ku.dk>

Supervisor: Anders Søgaard <soegaard@di.ku.dk>

Submitted on:

Abstract

Contents

Chapter 1

Introduction

1.1 Word embeddings are important

Word embedding is a representation for vocabulary in a document, more specifically, it represent vocabulary as vectors. Word embedding is able to capture information of how different words are related to each other based on the context in a document. The most polular and powerful technique to learn word embeddings is called Word2Vec which is a shallow neural network, and it was introduced by Mikolov et al. (2013).

1.2 Why cross embeddings is important

Cross-lingual word embeddings means mapping a source monolingual word embedding to another target embedding space in order to align both languages together in a same shared vector space. By Ruder et al. (2017), Cross-lingual word embeddings attract NLP researchers for two reasons:

- It makes computation of cross-lingual word similarities to enable.
- Knowledge can be transfer between different languages, even transfer resource-rich language to resource-insufficient language like English to Finnish.

Broadly speaking, there are two methods to learn cross-lingual embeddings, supervised method and unsupervised method. Conneau et al.(2017) reviewed the supervised method in their paper at beginning, then contribute a new unsupervised way to learn cross-lingual embeddings. The supervised method assumed that for a given pair of language embeddings, a small dictionary is given such that it would be used as a anchor point to align the two language embeddings. More specifically, assumed the embedding dimension of both embeddings is d, X and Y are matrices with size $d \times n$, and the known dictionary contains n pairs of words $\{x_i, y_i\}_{i \in 1,2,\cdots,n}$ for $x_i \in X$ and $y_i \in Y$, it can be used to learn a optimal linear transformation W between X and Y such that

$$W^* = \underset{W}{\text{argmin}} \, \|WX - Y\|_F \tag{1.1}$$

where W is a $d \times d$ matrix, and $\| \cdot \|_F$ denotes the Frobenius norm. After the optimal transformation obtained, for any source word s, the translation word t in target embeddings can be found by searching the nearest neighbor

$$t = \underset{t}{\text{argmax}} \cos(Wx_s, y_t).$$

The result of optimal W can be improved by orthogonalize it. The orthogonal improvement showed by Xing et al. (2015). Thus, the equation (1.1) can be reformed to the Orthogonal Procrustes problem, and solution would be reached by solving the singular value decomposition(SVD) of YX^T.

$$W^* = \underset{W}{\text{argmin}} \|WX - Y\|_F = UV^T,$$

Where $U\Sigma V^T = \text{SVD}(YX^T)$, the columns vector of U and V are orthonormal, and Σ is diagonal matrix which all diagonal elements are positive.

The unsupervised method which Conneau et al. proposed is like this:

- Assumed two vector spaces are approximately isomorphic, i.e exists an invertible linear mapping between them.
- Learn a rough approximation of mapping W by adversarial model. In particular, by generative adversarial networks (GANs).
- Applied the learned mapping W to find word pairs for the most frequent words, and then refine W by Procrusters. The refined W would be used to translate all words in the embedding space.
- In the end, applied W to translate words by searching nearest neighbor score, and the nearest neighbor algorithm they used is called cross-domain similarity local scaling(CSLS), which invented by Conneau et al..

1.3 What is GANs

Generative Adversarial Networks (GANs) are deep neural net architectures that consist of two networks, and GAN's developement is considered a revolutionary advancement in deep learning. The first GAN model proposed by Goodfellow (Goodfellow et al., 2014)[3], it is called vanilla GAN. The idea of GANs was inspired from game theory, in which two models are competing to each other, one is called generator, the other is discriminator, both models would become more and more robust during the training process. More specifically,

- The discriminator is a binary classifier denote as D which learns to recognize whether the input coming from real dataset or generator's output. Formally, D is to calculate the probability that whether the sample came from the generator G or the real input data for each iterations. Usually the label defined as 1 for real data and 0 for synthetic data.

- The generator denote as G which takes random noise from latent space, and output synthetic samples. The training purpose for generator is to approximate the distribution of real data as close as possible, at the same time the probability for lying the discriminator would increased too, and makes the discriminator to produce a high probability that treat the synthetic output from generator as real.

For measure the similarity between the distributions of generated samples and real samples, Jensen-Shannon (JS) divergence is used as the measurement instead of Kullback–Leibler (KL) divergence, because JS divergence is symmetric and more smoother than KL divergence.

GANs are able to generate data from scratch based on given sample. The main application area of GANs is computer vision, e.g., create anime characters (Jin et al. 2017) which could be applied by game developing or animation production company, high resolution images production based on given low resolution images (Ledig et al. 2016), and new video sequence generation (Vondrick et al. 2016) .

GANs can also be apply to other domains such as music generation (Fedus et al. 2018), text generationsn (Mogren. 2016), or even text to image synthesis (Reed at el. 2016) etc. These applications shows that GANs can be adapted immediately for commercial purpose.

Although vanilla GANs achieved great success, there are several disadvantages which makes vanilla GANs' training procedure unstable,

- Mode collapse frequently, which means the generator only learns limited distribution of real data.
- Lack of metric that can shows the training process, therefore the training is fully manual.
- Non- convergence, i.e. the model difficult to reach the Nash equilibrium due to oscillating change of gradients.
- Gradient vanished, which means the generator's weights stop updating due to the discriminator performing too well.
- Highly sensitive for tuning hyperparameters.

Several improved GANs tries to solve the above issues, e.g, Wasserstain GAN (Arjovsky et al. 2017), in which the Earth Mover (Wasserstein) Distance is applied to substitute the JS divergence as the new loss function to make gradient more stable due to the EM distance is much more smoother. Although the EM distance have nice properties, it is almost impossible to calculate, therefore, the Kantorovich-Rubinstein duality (Villani. 2008) algorithm is applied, where all weights in discriminator are forces to constrained to a 1-Lipschitz function, i.e., all weights in discriminator must be clipped in a bounded interval. However, weight clipping will bring new issues, even Arjovsky pointed out in the original Wasserstein GAN paper that it is a horrible method to enforce a Lipschitz constraint. By gradient penalty (Gulrajani, et al. 2017),

1.4 Several other improved GANs

Wasserstain GAN

ct-GAN (Wei, et al. 2018),

Dual discriminator

Vanilla GANs have the above problems, model collapse, lack of metric to measure the training procedure,. In this thesis, several improved version of GANs will be applied to the MUSE, these improved GANs theoretically solved the two main problems of vanilla GANs,

several questions: Can we apply these improved GANs to MUSE in order to get better performance?

Are these GANs only works for specific data set? or they can are generally applicable?

1.5 What should we do this this thesis

We focused on the unsupervised part of the MUSE in this thesis.

1: Try to reproduce they results Conneau et al. did, and use the reproduced results as benchmark.

2: Implement the improved GANs above, and compare the result with the benchmark, however, since the WGAN-GP has the best performance, we focused on it mostly.

3: Analysis the results we did.

Chapter 2

Vanilla GAN

The basic introduction of GANs already showed up in the previous chapter. This chapter is to mathematically introduce the mechanism of vanilla GAN, and why it is hard to train.

2.1 working mechanism

The training purpose for generator G is to make distribution of it's output more and more close to real data, the goal for discriminator D is to achieve a Nash equilibrium which impossible to distinguish whether it's input comes from real or generated dataset. Formally speaking, the task of discriminator D is to maximizing

$$\mathbb{E}_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}[\text{log}D(\boldsymbol{x})] \tag{1}$$

which means increasing the probability to identify samples coming from real data, as well as to reduce the probability $D(G(\boldsymbol{x}))$ to zero by maximizing

$$\mathbb{E}_{\boldsymbol{x}\sim p_g(\boldsymbol{x})}[\text{log}(1 - D(G(\boldsymbol{x})))].$$

Thus, the objective function of discriminator is

$$\max_D L(D) = \mathbb{E}_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}[\text{log}D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_g(\boldsymbol{z})}[\text{log}(1 - D(G(\boldsymbol{x})))]. \tag{2}$$

At the same time, the task for training generator G is to increasing the probability of output of D to assign label to fake data, therefore the objective function of generator is

$$\min_G L(G) = \mathbb{E}_{\boldsymbol{x}\sim p_g(\boldsymbol{x})}[\text{log}(1 - D(G(\boldsymbol{x})))].$$

Since D and G are playing minimax game, Combined both ideas, the final objective function is

$$\min_G \max_D L(G, D) = \mathbb{E}_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}[\text{log}D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_g(\boldsymbol{z})}[(1 - \text{log}D(G(\boldsymbol{x})))]. \tag{2}$$

The notation $p_r(\boldsymbol{x})$ means the probability distribution for real data, $p_z(\boldsymbol{x})$ denotes generated fake data.

Proposition 1: For fixed G, the optimal value for D is:

$$D^*(\boldsymbol{x}) = \frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}. \tag{3}$$

$Proof$:

Since the loss function in continuous probability distribution is

$$L(G, D) = \int_{\mathcal{X}} \Big(p_r(\boldsymbol{x})\log(D(\boldsymbol{x})) + p_{g(\boldsymbol{x})}\log(1 - D(\boldsymbol{x}))\Big)dx \tag{4}$$

the maximum of value $L(D, G)$ can be get by solving the function inside the integral. For convenient, Define the function inside the integral as $f(y) = p_r(x)\log y + p_g(x)\log\big(1 - y\big)$, and take the derivative of loss with respect to x and set it to zero get

$$\begin{aligned}
\frac{df(y)}{dy} &= p_r(x)\frac{1}{y} - p_g(x)\frac{1}{1-y} \\
&= \frac{p_r(x) - y\big(p_r(x) + p_g(x)\big)}{y(1-y)} \\
&= 0 \\
\Longrightarrow y &= \frac{p_r(x)}{p_r(x) + p_g(x)}
\end{aligned}$$

Apply the above to vector variable. Thus, the $D^*(\boldsymbol{x}) = \frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x})+p_g(\boldsymbol{x})} \in [0, 1]$. $\square$

By proposition 1 above, The equation (2) can be rewrite as

$$\begin{aligned}
L(G, D^*) &= \max_D L(G, D) \\
&= \mathbb{E}_{\boldsymbol{x}\sim p_r}[\,\log D^*(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim p_g}[(1 - \log D^*(\boldsymbol{x}))\,] \\
&= \mathbb{E}_{\boldsymbol{x}\sim p_r}\Big[\frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}\Big] + \mathbb{E}_{\boldsymbol{x}\sim p_g}\big[\frac{p_g(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}\big]
\end{aligned} \tag{5}$$

Theorem 1: The global minimum of $L(G, D^*)$ reached if and only if $p_r(\boldsymbol{x}) = p_g(\boldsymbol{x})$, at this point, $C(G) = -2\log 2$ and $D^*(\boldsymbol{x}) = 0.5$.

$Proof$:

By equation (3), if $p_r(\boldsymbol{x}) = p_g(\boldsymbol{x})$, then $D^*(\boldsymbol{x}) = 0.5$, substitute this value to equation (4) we have

$$\begin{aligned}\min_G L(G, D^*) =& \int_{\mathcal{X}} \Big(p_r(\boldsymbol{x})\log\frac{1}{2} + p_{g(\boldsymbol{x})}\log\frac{1}{2}\Big)dx\\ =&\log\frac{1}{2}\Big(\int_{\mathcal{X}} p_r(\boldsymbol{x})dx + \int_{\mathcal{X}} p_{g(\boldsymbol{x})}dx\Big)\\ =&2\log\frac{1}{2}\\ =&-2\log 2.\end{aligned}$$

to show $-2\log 2$ is the optimal value for $L(G, D^*)$, we need to prove they are equal to each other. Subtracting to each other and by applying proposition 1 get

$$\begin{aligned}L(G, D^*) - \big(-2\log 2\big) =& 2\log 2 + \int_{\mathcal{X}} \Big(p_r(\boldsymbol{x})\,\log\frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}\\ &+ p_g(\boldsymbol{x})\,\log\frac{p_g(\boldsymbol{x})}{p_r(x) + p_g(\boldsymbol{x})}\Big)dx\\ =&\Big(\log 2 + \int_{\mathcal{X}} p_r(\boldsymbol{x})\,\log\frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}dx\Big)\\ &+\Big(\log 2 + \int_{\mathcal{X}} p_g(\boldsymbol{x})\,\log\frac{p_g(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}dx\Big)\\ =&\Big(\int_{\mathcal{X}} p_r(\boldsymbol{x})\log 2\,dx + \int_{\mathcal{X}} p_r(\boldsymbol{x})\,\log\frac{p_r(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}dx\Big)\\ &+\Big(\int_{\mathcal{X}} p_g(\boldsymbol{x})\log 2\,dx + \int_{\mathcal{X}} p_g(\boldsymbol{x})\,\log\frac{p_g(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}dx\Big)\\ =&\int_{\mathcal{X}} p_r(\boldsymbol{x})\log\frac{2\cdot p_r(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}dx + \int_{\mathcal{X}} p_g(\boldsymbol{x})\log\frac{2\cdot p_g(\boldsymbol{x})}{p_r(\boldsymbol{x}) + p_g(\boldsymbol{x})}dx\\ =&KL\Big(p_r||\frac{p_r + p_g}{2}\Big) + KL\Big(p_g||\frac{p_r + p_g}{2}\Big)\\ =&2JS(p_r||p_g)\end{aligned}$$

Since the Jensen-Shannon divergence equals to zeros when the two probability distribution p_r and p_g equals everywhere, thus, the global optimal value for the loss is - 2 log2 and $D^*(\boldsymbol{x}) = 0.53227$. □

Below is the architecture draft of vanilla GAN and the training algorithm in the original paper (Goodfellow et all., 2014)[3].

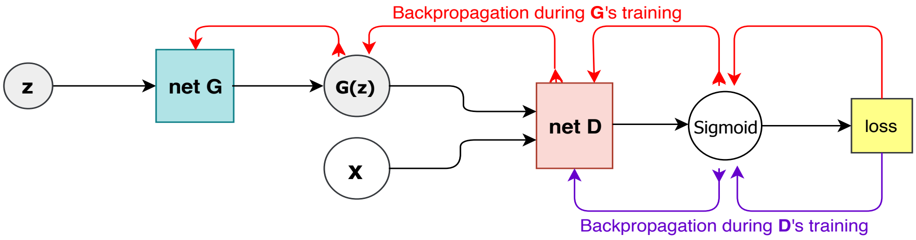

Backpropagation during G's training
z
net G
G(z)
x
net D
Sigmoid
loss
Backpropagation during D's training

Figure 1: The structure of Vanilla GAN

Algorithm 1: Minbatch training algorithm use stochastic gradient descent as optimizer, n is a hyperparameter which for each generator's iteration, trains discriminator D n iterations. The original paper used $n = 1$.

```
for i = 1, 2, ..., number of training iterations do
    for training discriminator $n$ iterations do
        Sample $k$ minibatch random noise from latent space distribution $p_g(\boldsymbol{z})$
        Sample $k$ minibatch input examples from data distribution $p_r(\boldsymbol{x})$
        Backpropagating all weights in $D$ by maximizing the loss:
            $\nabla_{W_D} \frac{1}{k} \sum_{i=1}^{k} \left[ \log D(\boldsymbol{x}^{(i)}) + \log \left( 1 - D\left( G(\boldsymbol{z}^{(i)}) \right) \right) \right].$
    end for
    Sample $k$ minibatch random noise from latent space distribution $p_g(\boldsymbol{z})$
    Backpropagating all weights in $G$ by minimizing the loss:
        $\nabla_{W_G} \frac{1}{k} \sum_{i=1}^{k} \log \left( 1 - D\left( G(\boldsymbol{z}^{(i)}) \right) \right).$
end for
```

However, Goodfellow et all point out that in practice, when training just start, the distribution of $\mathbb{P}_r$ and $\mathbb{P}_g$ may have huge difference, in this case, D can easily distinguish whether the data comes from training data or synthetic. Therefore, $\log(1 - D(G(\boldsymbol{z})))$ saturates. Maximize $\log D(G(\boldsymbol{z}))$can provide more robust gradients in the early of training instead of minimizing $\log(1 - D(G(\boldsymbol{z})))$.

2.2 Issues

By theorem 1, the minimum for discriminator's loss is

$$L(G, D^*) = 2JS(p_r || p_g) - 2\log 2$$

However, the discriminator's loss will converge to zero in practice . Numerically, $2\log 2 = 1.386294$, and this means that $JS(p_r || p_g)$ will converge to 0.693

instead of zero at which point the distributions of p_r and q_r are very similar.

By Arjovsky et al[19], the reason which cause the situation above happened is that either the supports of both distributions have supports which are disjoint, or they are discontinuous, in other words, probability density function not exist. Arjovsky et al clarified that the term continuous means absolutely continuous, for an absolutely random variable, it have has property that $P(X \in B) = 0$ in which B has 0 Lebesgue measure. Whereas a random variable is continuous if $P(X = x) = 0 \ \forall x \in X, x$ are points. Absolutely continuous implies continuous since the Lebesgue measure of points is 0, but not vise versa. Movrover, If a random variable has support on low dimensional manifold, it is not absolutely continuous. Narayanan et al. (2010) claimed that the support of distribution of $\mathbb{P}_r$ lies on low dimensional manifold.

For $\mathbb{P}_g$, if dim($\mathcal{Z}$) $<$ dim($\mathcal{X}$), then $\mathbb{P}_g$ is not continuous, and it formalized in the following lemma.

Lemma 1: [Lemma 1. (Arjovsky et al. 2017)] Assume $g : \mathcal{Z} \mapsto \mathcal{X}$ be a neural network, which is a function composed by affine transformations with non-linear activation functions, then a countable union of manifolds which contains $g(\mathcal{Z})$ would have dimensionality less than or equal to dim($\mathcal{Z}$). Thus, the measure of $g(\mathcal{Z})$ would be zero in $\mathcal{X}$ if dim($\mathcal{Z}$) $<$ dim($\mathcal{X}$).

The next theorem state that if the supports of two distributions $\mathbb{P}_r$ and $\mathbb{P}_g$ are disjoint, there exist a

Lemma 2: (Smooth Urysohn's Lemma) [Need citation]
Let $\mathcal{M}$ and $\mathcal{S}$ be two compact and disjoint subsets in a normal topological space $(\mathcal{X}, \mathcal{T})$, then there exists a function $f : \mathcal{X} \mapsto [0, 1]$ which is smooth such that $f(x) = 0 \, \forall x \in \mathcal{S}$ and $f(x) = 1 \, \forall x \in \mathcal{M}$.

the claims are still highly debatable.

The next theorem state that if the supports of two distributions $\mathbb{P}_r$ and $\mathbb{P}_g$ are disjoint, a perfect discriminator can be reached, along with vanishing gradients. In this case, therefore, when D trained too well, the weights update of G will finish, and make G stop to approximate.

Theorem 3: [Theorem 2.1 (Arjovsky et al. 2017)]
Assume the supports of $\mathbb{P}_r$ and $\mathbb{P}_g$ included on disjoint compact subsets $\mathcal{M}$ and $\mathcal{S}$ respectively, then there exists a complete accurate smooth discriminator $D^* : \mathcal{X} \mapsto [0, 1]$ which have 0 error, plus $\nabla_{\boldsymbol{x}} D^*(\boldsymbol{x}) = 0, \ \forall \boldsymbol{x} \in \mathcal{M} \cup \mathcal{S}$.

$Proof$:

By assumption, $\mathcal{M}$ and $\mathcal{S}$ are disjoint and compact. Supposed $d(\mathcal{M}, \mathbb{S}) = \epsilon > 0$

is the distance between $\mathcal{M}$ and $\mathcal{S}$, then define another two sets

$$\mathcal{M}' = \{\boldsymbol{x} : d(\boldsymbol{x}, \mathcal{M}) \leq \frac{\epsilon}{3}\}$$
$$\mathcal{S}' = \{\boldsymbol{x} : d(\boldsymbol{x}, \mathcal{S}) \leq \frac{\epsilon}{3}\}$$

therefore $\mathcal{M}'$ and $\mathcal{S}'$ are also both disjoint and compact. Thus, by applying Smooth Urysohn's Lemma, exists a continuous function $D^* : \mathcal{X} \mapsto [0, 1]$ such that $D^*\big|_{\mathcal{S}'} \equiv 0$ and $D^*\big|_{\mathcal{M}'} \equiv 1$. Since $\forall \boldsymbol{x} \in \text{supp}(D^*\big|_{\mathbb{P}_r}) = \{\boldsymbol{x} \in \mathbb{P}_r \big| D^* \neq 0\}$, $\log D^*(\boldsymbol{x}) = 0$, and $\log \big(1 - D^*(\boldsymbol{x})\big) = 0\ \forall \boldsymbol{x} \in \text{supp}(D^*\big|_{\mathbb{P}_g})$, therefore discriminator has zero error and reached complete optimal. In addition, for proving $D^*(\boldsymbol{x}) = 0$, first define $\boldsymbol{x} \in \mathcal{M} \cup \mathcal{S}$, then for $\boldsymbol{x} \in \mathcal{M}$, exist an open ball $\mathcal{B}(\boldsymbol{x}, \frac{\epsilon}{3}) = \mathcal{B}'$ such that $D^*\big|_{\mathcal{B}'}$ is constant. Thus $\nabla_{\boldsymbol{x}} D^* \equiv 0$. For $\boldsymbol{x} \in \mathcal{S}$, following the same procedure will done the proof. □

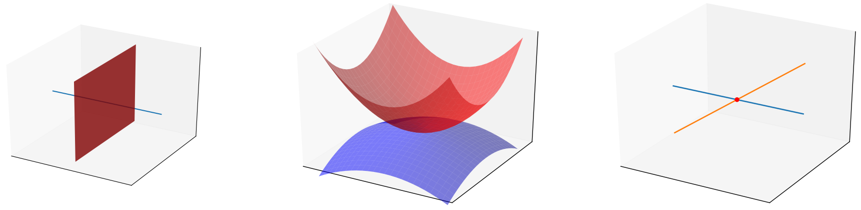
If two manifolds align perfectly on a space, no perfect discriminator can be learned. However, the probability for

Definition 1: (Transversal intersection need citation) Assume $\mathcal{M}$ and $\mathcal{S}$ are two submanifolds of $\mathcal{F}$, and $\mathcal{F} = \mathbb{R}^n$. If $\forall p \in \mathcal{M} \cap \mathcal{S}$

$$T_p\mathcal{M} + T_p\mathcal{S} = T_pF$$

then we say that $\mathcal{M}$ and $\mathcal{S}$ intersect transversally.

$T_p\mathcal{M}$ means the tangent space of $\mathcal{M}$ at point p. Moreover, if two submanifolds does not have intersection, then they are transversal.



(a) transverse

(b) non-transverse

(c) non-transverse

Figure 1: examples of transversal intersection

Definition 2: (Perfect align) [Definition 2.2 (Arjovsky et al. 2017)] Assume $\mathcal{M}$ and $\mathcal{S}$ are two manifolds without boundary. We say $\mathcal{M}$ and $\mathcal{S}$ perfectly align if exist $\boldsymbol{x} \in \mathcal{M} \cap \mathcal{S}$ such that $\mathcal{M}$ and $\mathcal{S}$ are non-transversal at $\boldsymbol{x}$.

However, in practice, we can assume that two manifolds can never be perfect align.

Lemma 2: [Lemma 2. (Arjovsky et al. 2017)]

Assume $\mathcal{M}$ and $\mathcal{S}$ are two regular submanifolds of $\mathbb{R}^n$, and they are not full dimension. Supposed ϵ and ϵ' are two independent continuous random variables, define two perturbed manifolds $\hat{\mathcal{M}} = \mathcal{M} + \epsilon$ and $\hat{\mathcal{S}} = \mathcal{S} + \epsilon'$. Then

$$\mathbb{P}_{\epsilon,\epsilon'}\left(\hat{\mathcal{M}} \text{ and } \hat{\mathcal{S}} \text{ not perfectlyh align }\right) = 1$$

This lemma is the preparation for the later theorem 2.2, it says any small perturbations can make two low dimension manifolds become non-perfect align, i.e., transversally intersect.

Lemma 3: [Lemma 3. (Arjovsky et al. 2017)]
Assume $\mathcal{M}$ and $\mathcal{S}$ are two non-full dimension and non-perfect align regular submanifolds of $\mathbb{R}^n$. Let $\mathcal{M} \cap \mathcal{S} = \mathcal{P}$.

- If $\mathcal{M}$ and $\mathcal{S}$ have boundary, then $\mathcal{P}$ is a union of no more than four strictly low dimensional manifolds.
- If $\mathcal{M}$ and $\mathcal{S}$ without boundary. then $\mathcal{P}$ is still a manifold which has strictly lower dimension than $\mathcal{M}$ or $\mathcal{S}$.

$\mathcal{P}$ has measure zero in $\mathcal{M}$ and $\mathcal{S}$ in both cases.

Theorem 4: [Theorem 2.2 (Arjovsky et al. 2017)]
Assume $\mathbb{P}_r$ and $\mathbb{P}_g$ are two distributions, their supports contained respectively in two closed lower dimensional manifolds $\mathcal{M}$ and $\mathcal{S}$ which are non-perfect align. In addition, suppose $\mathbb{P}_r$ and $\mathbb{P}_g$ are continuous, which means that for any set $A \in \mathcal{M}$ with zero measure implies $\mathbb{P}_r(A) = 0$, similar for $\mathbb{P}_g$. Then there is an full accuracy optimal discriminator $D^* : \mathcal{X} \mapsto [0, 1]$ which can distinguish $\mathbb{P}_r$ and $\mathbb{P}_g$. Moreover, $\forall \boldsymbol{x} \in \mathcal{M}$ or $\forall \boldsymbol{x} \in \mathcal{S}$, $\nabla_{\boldsymbol{x}} D^*(\boldsymbol{x}) = 0$ and D^* is smooth in a neighborhood.

Theorem 5: [Theorem 2.3 (Arjovsky et al. 2017)]
Assume $\mathbb{P}_r$ and $\mathbb{P}_g$ are two distributions, their supports contained respectively in two lower dimensional manifolds $\mathcal{M}$ and $\mathcal{S}$ which are non-perfect align. Further more, assume $\mathbb{P}_r$ and $\mathbb{P}_g$ are continuous in their manifolds. Then

- $JSD(\mathbb{P}_r \| \mathbb{P}_g)$ = log 2
- $KL(\mathbb{P}_r \| \mathbb{P}_g) = +\infty$
- $KL(\mathbb{P}_g \| \mathbb{P}_r) = +\infty$

Theorem 6: [Theorem 2.4 (Arjovsky et al. 2017)]
Assume $g_\theta : Z \mapsto \mathcal{X}$ is a differentiable function which induced the distribution $\mathbb{P}_g$, $\mathbb{P}_r$ be the input data distribution, D is a differentiable discriminator. If the conditions in theorem 2.1 and 2.2 are satisfied, $\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[J_\theta g_\theta(z)\|_2^2\right] \leq M^2$ and $\|D - D^*\| < \epsilon$, then

$$\left\|\nabla_\theta \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\left[\log\left(1 - D(g_\theta(\boldsymbol{z}))\right)\right]\right\|_2 < M\frac{\epsilon}{1-\epsilon}$$

$Proof$:

Since the theorem 2.1 and 2.1 satisfied, $\nabla_{\boldsymbol{x}} D^*(\boldsymbol{x}) = 0$. Therefore, by Jensen's inequality and chain rule

$$\begin{aligned}
\left\|\nabla_\theta \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\big[\log\big(1 - D(g_\theta(\boldsymbol{z}))\big)\big]\right\|_2^2 &\le \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\Big[\frac{\|\nabla_\theta D\big(g_\theta(\boldsymbol{z})\big)\|_2^2}{|1 - D\big(g_\theta(\boldsymbol{z})\big)|^2}\Big] \\
&= \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\Big[\frac{\|J_\theta g_\theta\big(\boldsymbol{z}\big)\nabla_{\boldsymbol{x}} D\big(g_\theta(\boldsymbol{z})\big)\|_2^2}{|1 - D\big(g_\theta(\boldsymbol{z})\big)|^2}\Big] \\
&\le \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\Big[\frac{\|\nabla_{\boldsymbol{x}} D\big(g_\theta(\boldsymbol{z})\big)\|_2^2 \|J_\theta g_\theta\big(\boldsymbol{z}\big)\|_2^2}{|1 - D\big(g_\theta(\boldsymbol{z})\big)|^2}\Big] \\
&\le \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\Big[\frac{\big(\|\nabla_{\boldsymbol{x}} D^*\big(g_\theta(\boldsymbol{z})\big)\|_2 + \epsilon\big)^2 \|J_\theta g_\theta\big(\boldsymbol{z}\big)\|_2^2}{\big(|1 - D^*\big(g_\theta(\boldsymbol{z})\big)| - \epsilon\big)^2}\Big] \\
&= \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\Big[\frac{\epsilon^2 \|J_\theta g_\theta\big(\boldsymbol{z}\big)\|_2^2}{\big(1 - \epsilon\big)^2}\Big] \\
&\le M^2 \frac{\epsilon^2}{(1-\epsilon)^2}
\end{aligned}$$

After square root for both side, get the final result

$$\left\|\nabla_\theta \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\big[\log\big(1 - D(g_\theta(\boldsymbol{z}))\big)\big]\right\|_2 < M \frac{\epsilon}{1-\epsilon}$$

□

Corollary 1: [Corollary 2.1 (Arjovsky et al. 2017)]
Assume all conditions from theorem 2.4 are satisfied, then

$$\lim_{\|D - D^*\| \to 0} \nabla_\theta \mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\big[\log\big(1 - D(g_\theta(\boldsymbol{z}))\big)\big] = 0$$

Theorem and corollary says when $D \to D^*$, the gradient with respect to G will vanish, therefore, in this situation, generator stop learning the input distribution .

In practice, the alternative loss function log$D(G(\boldsymbol{z}))$ does not cause gradient vanish, but still it's gradient suffer from unstable update.

Theorem 7: [Theorem 2.5 (Arjovsky et al. 2017)]
Assume $\mathbb{P}_r$ and $\mathbb{P}_g$ are two distributions with density function P_r and P_g respectively. $D^* = \frac{P_r}{P_{g_{\theta_0}} + P_r}$ is the optimal discriminator with a fixed value θ_0. Then

$$\mathbb{E}_{\boldsymbol{z}\sim p(\boldsymbol{z})}\big[- \nabla_\theta \log D^*(g_\theta(z))|_{\theta=\theta_0}\big]$$

The gradient of the new alternative loss function have a inverse KL-divergence and minus JS-divergence between $\mathbb{P}_r$ and $\mathbb{P}_g$. The inverse KL-term can penalize largely different samples by assigns high cost for them, but low cost for mode collapse. Since JS divergence is symmetric, therefore it can not change the problem.

Chapter 3

Wasserstein GAN

The way how vanilla GAN works and why it's hard to train already showed up in the previous chapter. Arjovsky et al. (2017) theoretically explored the drawbacks of Vanilla GAN and apply the Earth Mover Distance(EMD) to introduce a new GAN called Wasserstein GAN which tries to solve them, e.g., mode collapse, and lack of interpretable metric that tells us training progress. However, even Wasserstein GAN have better performance than vanilla GAN in image generation, it still have some flaws such weight clipping.

3.1 Several probability metrics

There are several formula to measure the similarity between two probability distributions over the same random variable x. Assumed $\mathbb{P}_r(\boldsymbol{x})$ and $\mathbb{P}_g(\boldsymbol{x})$ are two distributions.

Kullback-Leibler (KL) divergence (continuous form):

$$KL(\mathbb{P}_r||\mathbb{P}_g) = \int_{\mathcal{X}} \log\Big(\frac{P_r(\boldsymbol{x})}{P_g(\boldsymbol{x})}\Big) P_r(\boldsymbol{x}) dx,$$

it's lower bounded by 0. KL Divergence reaches minimum zero while $P_r(x) = P_g(x)$ for all $x \in \mathcal{X}$, it is asymmetric, . The discrete form of KL divergence is :

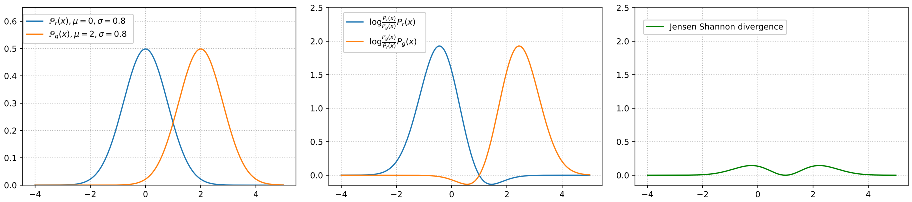
$$\begin{aligned} KL(\mathbb{P}_r||\mathbb{P}_g) =& H(p) - H(p,q) \\ =& \sum_x \log\Big(\frac{P_r(\boldsymbol{x})}{P_g(\boldsymbol{x})}\Big) P_r(x) \\ =& -\sum_x \log\Big(\frac{P_g(\boldsymbol{x})}{P_r(\boldsymbol{x})}\Big) P_r(x), \end{aligned}$$

where $H(p, q)$ is cross entropy between p and q. KL divergence is asymmetric.

Jensen-Shannon(JS) divergence

$$JS(\mathbb{P}_r, \mathbb{P}_g) = \frac{1}{2} KL(\mathbb{P}_r||\mathbb{P}_m) + \frac{1}{2} KL(\mathbb{P}_g||\mathbb{P}_m)$$

where $\mathbb{P}_m = \frac{\mathbb{P}_r+\mathbb{P}_g}{2}$, the JS divergence is symmetric, and bounded in $[0, 1]$. It is symmetric, i.e. $JS(\mathbb{P}_r, \mathbb{P}_g) = JS(\mathbb{P}_g, \mathbb{P}_r)$.



(a) $\mathbb{P}_r(\mathcal{X})$ and $\mathbb{P}_g(\mathcal{X})$ (b)$KL(\mathbb{P}_r||\mathbb{P}_g)$ and $KL(\mathbb{P}_g||\mathbb{P}_r)$ (c) Jensen Shannon divergence

Figure 1: Two different normal distributions with corresponding KL and JSD

Earth Mover Distance(EMD):

The last measure is The Earth Mover's Distance (EMD), also called Wasserstein Distance. Informally, assumed there are two different distribution piles of dirt $\mathbb{P}$ and $\mathbb{Q}$ with same total amount, the EMD can be interpreted as the minimal cost for moving one distribution piles to another to make both $\mathbb{P}$ and $\mathbb{Q}$ have the same distribution. Thus we can calculate EMD as the sum of work flow for moving piles from $\mathbb{P}$ to $\mathbb{Q}$, for each move, the mass times it distance. Sometimes there are lots of ways for move one distribution to another therefore to find the way with minimal cost is an optimization problem.

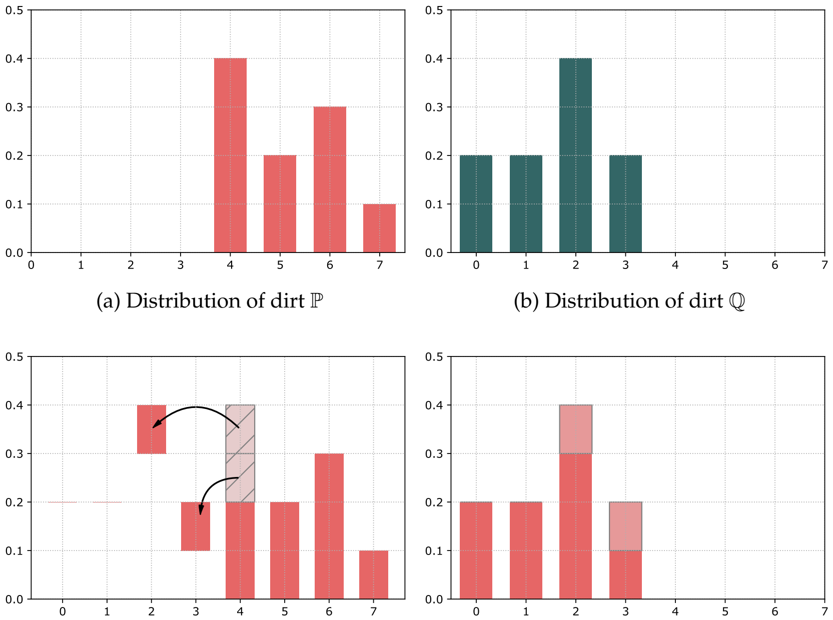
Definition of discrete case:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \sum_{\boldsymbol{x},\boldsymbol{y}} \|\boldsymbol{x} - \boldsymbol{y}\| \gamma(\boldsymbol{x}, \boldsymbol{y}) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y}) \sim \gamma} \left[\|\boldsymbol{x} - \boldsymbol{y}\|\right]$$

where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ is the set of all combination of joint distributions between $\mathbb{P}_r$ and $\mathbb{P}_g$. Since γ means the percentage of earth should be moved from $\boldsymbol{x}$ to $\boldsymbol{y}$, thus $\mathbb{P}_g(\boldsymbol{y}) = \sum_{\boldsymbol{x}} \gamma(\boldsymbol{x}, \boldsymbol{y})$ and $\mathbb{P}_r(\boldsymbol{x}) = \sum_{\boldsymbol{y}} \gamma(\boldsymbol{x}, \boldsymbol{y})$, which means $\mathbb{P}_g(\boldsymbol{y})$ and $\mathbb{P}_r(\boldsymbol{x})$ are marginal distributions of $\gamma(\boldsymbol{x}, \boldsymbol{y})$ respectively.

Here is a simple example of how Earth Mover's Distance works, suppose X and Y are two different discrete random variables over the spaces $\mathbb{P}$ and $\mathbb{Q}$, then the EMD from $\mathbb{P}$ to $\mathbb{Q}$ is

$$\begin{aligned}
\text{EMD}(\mathbb{P}, \mathbb{Q}) =& 0.1 * |4 - 3| + 0.1 * |4 - 2| + 0.2 * |4 - 0| \\
&+ 0.2 * |5 - 1| + 0.3 * |6 - 2| + 0.1 * |7 - 3| \\
=& 3.5
\end{aligned}$$



(a) Distribution of dirt $\mathbb{P}$

(b) Distribution of dirt $\mathbb{Q}$

(c) Moving dirt $\mathbb{P}$ to $\mathbb{Q}$

(d) transported $\mathbb{P}$

Figure 1: Transporting discrete probability distributions $\mathbb{P}$ to $\mathbb{Q}$

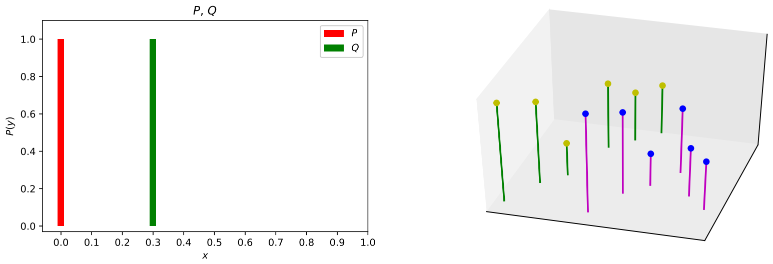
3.2 Why Wasserstain GAN is better

Here is an example of why using Wasserstein:

Let Y be the random variable $Y \sim U[0,1]$, $\mathbb{P}$ and $\mathbb{Q}$ are two probability distributions, where $\mathbb{P}$ is the distribution on $(0, Y) \in \mathbb{R}^2$, $\mathbb{Q}$ is $(\theta, Y) \in \mathbb{R}^2$ for $\theta \in [0,1]$. If $\theta \neq 0$ then we have:

$$
\begin{aligned}
& KL(\mathbb{P}||\mathbb{Q}) = \sum_{x=0, Y\sim U[0,1]} 1 \times \ln\frac{1}{0} = \infty \\
& KL(\mathbb{Q}||\mathbb{P}) = \sum_{x=\theta, Y\sim U[0,1]} 1 \times \ln\frac{1}{0} = \infty \\
& JS(\mathbb{P}||\mathbb{Q}) = \frac{1}{2}KL\Big(\mathbb{P}||\frac{\mathbb{P}+\mathbb{Q}}{2}\Big) + \frac{1}{2}KL\Big(\mathbb{Q}||\frac{\mathbb{Q}+\mathbb{P}}{2}\Big) \\
=& \frac{1}{2}\Big(\sum_{x=0, Y\sim U[0,1]} 1 \times \ln\frac{1}{0.5} + \sum_{x=\theta, Y\sim U[0,1]} 1 \times \ln\frac{1}{0.5}\Big) \\
=& \ln 2 \\
& W(\mathbb{P},\mathbb{Q}) = |\theta|
\end{aligned}
$$

If $\theta = 0$, then $\mathbb{P}$ and $\mathbb{Q}$ are completely overlapped. This indicates that in most cases, KL divergence is impossible to calculate when two distributions are discrete.



2d

3d

Figure 1: 2d and 3d plot of non-overlapped probability distributions $\mathbb{P}$ and $\mathbb{Q}$

Lebesgue's Dominated Convergence Theorem.
Let $\{f_n\}$ be a sequence of real-valued measurable functions on a measure space (S, Σ, μ). Suppose that the sequence converges pointwise to a function f

$$\left|f_n(x)\right| \leq f(x)$$

for all the index number $n \in \mathbb{N}^+$ and $x \in S$, then f is integrable and

$$\lim_{n \to \infty} \int_S \left|f_n - f\right| d\mu = 0$$

Radamacher's theorem
If U is an open subset in $\mathbb{R}^n$, $f : U \mapsto \mathbb{R}^m$ is Lipschitz continuous, then f is differentiable almost everywhere.

Assumption 1
Let $g : \mathcal{Z} \times \mathbb{R}^d \mapsto \mathcal{X}$ be locally Lipschitz between finite dimentional vector spaces, $g_\theta(z)$ is denoted as evaluation on corrdinates (z, θ). We call g satisfies assumption 1 for a certain probability distribution $\mathbb{P}$ over $\mathcal{Z}$ if there are local Lipschitz constants $K(\theta, z)$ such that:

$$\mathbb{E}_{z \sim \mathbb{P}}\left[K(\theta, z)\right] < \infty$$

Theorem 1
Let $\mathbb{P}_r$ be a fixed distribution over $\mathcal{X}$. Let Z be a random variable (e.g Gaussian) over another space $\mathcal{Z}$. Let $g : \mathcal{Z} \times \mathbb{R}^d \mapsto \mathcal{X}$ be a function, that will be denoted $g_\theta(\boldsymbol{z})$ with $\boldsymbol{z}$ the first coordinate and $\boldsymbol{\theta}$ the second. Let $\mathbb{P}_\theta$ denote the distribution of $g_\theta(Z)$. Then:

1. If g is continuous in $\boldsymbol{\theta}$, then $W(\mathbb{P}_r, \mathbb{P}_\theta)$ is also continuous.
2. If g is locally Lipschitz and satisfies regularity assumption 1, then $W(\mathbb{P}_r, \mathbb{P}_\theta)$ is continuous everywhere, and differentiable almost everywhere.

Proof.

1.

The goal is to show

$$\begin{aligned} &\lim_{\theta_1 \to \theta_2} \|g_{\theta_1}(\boldsymbol{z}) - g_{\theta_2}(\boldsymbol{z})\| = 0 \\ \Longrightarrow &\lim_{\theta_1 \to \theta_2} \|W(\mathbb{P}_r, \mathbb{P}_{\theta_1}) - W(\mathbb{P}_r, \mathbb{P}_{\theta_2})\| = 0 \end{aligned}$$

Assume θ_1 and θ_2 are two parameter vectors in $\mathbb{R}^d$. By the definition of the Wasserstein distance,

$$\begin{aligned} W(\mathbb{P}_{\theta_1}, \mathbb{P}_{\theta_2}) &\leq \mathbb{E}_{(a,b)\sim\gamma}\left[\|x - y\|\right] \\ &= \mathbb{E}_{(a,b)\sim\gamma}\left[\|g_{\theta_1(z)} - g_{\theta_2(z)}\|\right] \\ &= \int_{\mathcal{X}\times\mathcal{X}} \left[\|g_{\theta_1(z)} - g_{\theta_2(z)}\|\right] d\gamma \end{aligned} \tag{1}$$

If g is continuous, by the property of continuity,

$$\begin{aligned} &\lim_{\theta_1 \to \theta_2} g_{\theta_1}(z) = g_{\theta_2}(z) \\ \Longrightarrow &\|g_{\theta_1}(z) - g_{\theta_2}(z)\| = 0. \end{aligned}$$

Since we assumed that $\mathcal{X}$ is compact, which means closed and bounded, therefore exists a constant M such that $\|g_{\theta_1}(z) - g_{\theta_2}(z)\| \leq M$ uniformly for all $\theta \in \mathbb{R}^d$ and z. By the Lebesgue's dominated convergence theorem and (1),

$$\lim_{\theta_1 \to \theta_2} \int_{\mathcal{X}\times\mathcal{X}} \left[\|g_{\theta_1(z)} - g_{\theta_2(z)}\|\right] d\gamma = \lim_{\theta_1 \to \theta_2} \mathbb{E}_z\left[\|g_{\theta_1}(z) - g_{\theta_2}\|\right] = 0$$

by the triangle inequality:

$$\Big\| W\big(\mathbb{P}_r, \mathbb{P}_{\theta_1}\big) - W\big(\mathbb{P}_r, \mathbb{P}_{\theta_2}\big) \Big\| \leq W\big(\mathbb{P}_{\theta_1}, \mathbb{P}_{\theta_2}\big)$$

Therefore if $\theta_1 \to \theta_2$,

$$\lim_{\theta_1 \to \theta_2} W\big(\mathbb{P}_{\theta_1}, \mathbb{P}_{\theta_2}\big) = 0$$

The continuity of $W(\mathbb{P}_r, \mathbb{P}_\theta)$ is proved.
For proving 2, assumed g is Lipschitz locally, that is, for a given point (z, θ) in the domain of g, exists an open set $(z, \theta) \in U$ and a constant $K(z, \theta)$ such that $\forall (z', \theta') \in U$ satisfied the property,

$$\big\|g_\theta(z) - g_{\theta'}(z')\big\| \leq K(z,\theta)\big\|(\theta, z) - (\theta', z')\big\| \leq K(z,\theta)\big(\|\theta - \theta'\| + \|z - z'\|\big).$$

After taking expectation for both sides of the above equation and set $z = z'$,

$$\mathbb{E}_{z\sim\mathbb{P}_\theta}\big[\|g_\theta(z) - g_{\theta'}(z)\|\big] \leq \mathbb{E}_{z\sim\mathbb{P}_\theta}\big[K(z,\theta)\big]\big\|\theta - \theta'\big\| \qquad \forall\big(z, \theta'\big) \in U$$

define a new set

$$U_\theta := \big\{\theta' \big| (\theta', z) \in U\big\}.$$

Since U is open set, thus U_θ as well. Based on the above proof, by assumption 1 and define $K(\theta) = \mathbb{E}_{z\sim\mathbb{P}_\theta}\big[K(z,\theta)\big]$ we have:

$$\begin{aligned}\Big\|W\big(\mathbb{P}_r, \mathbb{P}_{\theta_1}\big) - W\big(\mathbb{P}_r, \mathbb{P}_{\theta_2}\big)\Big\| \leq & W\big(\mathbb{P}_{\theta_1}, \mathbb{P}_{\theta_2}\big) \\ \leq & \mathbb{E}_{z\sim\mathbb{P}_\theta}\big[\|g_\theta(z) - g_{\theta'}(z')\|\big] \\ \leq & K(\theta)\big\|\theta - \theta'\big\| \qquad\qquad \forall \theta' \in U_\theta\end{aligned}$$

this implies that $W\big(\mathbb{P}_r, \mathbb{P}_{\theta_1}\big)$ is locally Lipschitz, and also continuous everywhere. By Radamacher's theorem, $W\big(\mathbb{P}_r, \mathbb{P}_{\theta_1}\big)$ muse be differentiable almost everywhere. □

There is a corollary guarantees that Wasserstain Distance can be used as loss function in neural networks.

Corollary 1
If g_θ is a feedforward neural network with $\boldsymbol{\theta}$ as it's parameters, that is, g_θ is a function composed by affine transformations and non-linear activation functions which are smooth Lipschitz continuous, and $\mathbb{E}_{z\sim p(z)}\big[\|z\|\big] < \infty$. Then the assumption 1 is satisfied and therefore $W(\mathbb{P}_r, \mathbb{P}_\theta)$ is continuous everywhere and differentiable almost everywhere.

3.3 Wasserstein GAN

The solution of Wasserstein distance' infiumum form in (1) is hard to find. However, solving (1) is equivalent to solve the Kantorovich - Rubinstein duality (Villani. 2008)

$$W(\mathbb{P}_r, \mathbb{P}_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}_r(\boldsymbol{x})} - \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}_g(\boldsymbol{x})} \tag{5}$$

where all functions of $f : \mathcal{X} \mapsto \mathbb{R}$ are 1-Lipschitz. If substitute $\|f\|_L \leq 1$ to $\|f\|_L \leq K$, then the left side of the above equations would be $K \cdot W(\mathbb{P}_r, \mathbb{P}_\theta)$, and we are still in the same optimization problem. Thus, we could solving the Wasserstein distance by maximizing

$$\max_{w \in \mathcal{W}} \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}_r(\boldsymbol{x})} \left[f_w(\boldsymbol{x}) \right] - \mathbb{E}_{\boldsymbol{x} \sim \mathbb{P}_r(\boldsymbol{x})} \left[f_w(g(\boldsymbol{z})) \right]$$

Where w are parameters for family of functions $\{f_w\}_{w \in \mathcal{W}}$, and w belong to a compact space $\mathcal{W}$, this implies that all f_w are K- Lipschitz functions depend only on $\mathcal{W}$. In order to implement that, Arjovsky et al. proposed to regulate all weights w of discriminator's neural network, e.g., $w \in [-0.01, 0.01]$ after update all weights. However, Arjovsky et al. also pointed out that clip weight is a bad way to implement Lipschitz constraint. Since if the clipping bound is too large, it would make lot of extra iterations for all weights w to converge their limit. On the contrary, if the clipping bound is too small, it would cause vanishing gradients. But still, since the Wasserstein distance is differentiable almost everywhere, ideally, the critic could be trained until optimality.

In addition, the backpropagation of G describe in the theorem below

Theorem 3 [Theorem 3 (Arjovsky et al. 2017)]
Assume $\mathbb{P}_r$ is a distribution, and $\mathbb{P}_g$ is another distribution over $g_\theta(Z)$ with Z as random variable, $g_\theta(Z)$ satisfied assumption 1, then there exists a solution $f : \mathcal{X} \mapsto \mathbb{R}$ for (5). Moreover,

$$\nabla_\theta W(\mathbb{P}_r, \mathbb{P}_\theta) = -\mathbb{E}_{\boldsymbol{z} \sim p_g(z)} \left[\nabla_\theta f(g_\theta(\boldsymbol{z})) \right].$$

With the above theoretical exploration, the final WGAN algorithm they present is like below

Algorithm 2: WGAN algorithm with default hyperparameters proposed by Arjovsky et al. Both D and G are use RMSprop as optimizer, with learn rate $\lambda = 0.00005$, weight clip $c = 0.01$, $n = 5$, batch size $k = 64$

```
while before $\theta$ converged do
    for training discriminator $n$ iterations do
        Sample $k$ minibatch random noise from latent space distribution $p_g(\boldsymbol{z})$
        Sample $k$ minibatch input examples from data distribution $p_r(\boldsymbol{x})$
        Backpropagating all weights in $D$ by maximizing and update the loss:
            $$g_{w_D} \leftarrow \nabla_{w_D} \frac{1}{k} \sum_{i=1}^{k} \Big[ f_w(\boldsymbol{x}^{(\boldsymbol{i})}) - f_w\big(g(\boldsymbol{z}^{(\boldsymbol{i})})\big) \Big].$$
        $w_D \leftarrow w_D + \lambda\cdot$ RMSProp$(w_D, g_{w_D})$
        $w_D \leftarrow$ clip$(w_D, [-c, c])\ \forall w_D \notin [-c, c]$
    end for
    Sample $k$ minibatch random noise from latent space distribution $p_g(\boldsymbol{z})$
    Backpropagating all weights in $G$ by maximizing and update the loss:
        $$g_{W_G} \leftarrow -\nabla_{W_G} \frac{1}{k} \sum_{i=1}^{k} \Big[ f_w\big(g(\boldsymbol{z}^{(\boldsymbol{i})})\big) \Big]$$
    $w_G \leftarrow w_G - \lambda\cdot$ RMSProp$(w_G, g_{w_G})$
end while
```

The loss function is an approximation of Wasserstain distance

Chapter 4

Several improved GANs

4.1 Wasserstein GAN with Gradient Penalty (WGAN-GP)

In the previous chapter, we know that in Wasserstein GAN, Arjovsky et al introduced the new loss functions for D and G respectively

- $L(D) = -\mathbb{E}_{\boldsymbol{x}\sim p_r(\boldsymbol{x})}\big[D(\boldsymbol{x})\big] + \mathbb{E}_{\boldsymbol{x}\sim p_g(\boldsymbol{z})}\big[D(G(\boldsymbol{z}))\big]$
- $L(G) = \mathbb{E}_{\boldsymbol{x}\sim p_g(\boldsymbol{z})}\big[D(G(\boldsymbol{z}))\big]$

By the Kantorovich-Rubinstein duality, the output of discriminator much be enforced to a K-Lipschitz constraint. In other words, the L2 norm of D's gradient must be bounded by a constant K

$$\big\|\nabla_{\boldsymbol{x}} D(\boldsymbol{x})\big\| \leq K$$

In the previous chapter, Arjovsky et al introduced weight clipping

$$w \leftarrow \text{clip}(w, [-c, c])$$

to implement Lipschitz constrain, where c is the hyperparameter. However, as Gulrajani et al point out in their paper, this method have two problems:

1. gradient can easily vanish or explode
2. most of weights are distributed on the clipping boundary, i.e., either equals to $-c$ or c.

In the first problem, since discriminator usually are deep neural networks, therefore, by the chain rule, if clipping constant c is being set relatively small, the backpropagation can cause gradient vanish; if c being set relatively big, gradient can easily explode. Instead of applying weight clipping, Gulrajani et al showed on Swiss Roll dataset (Figure 10) that the gradient penalty term can make gradient stably through each layer.

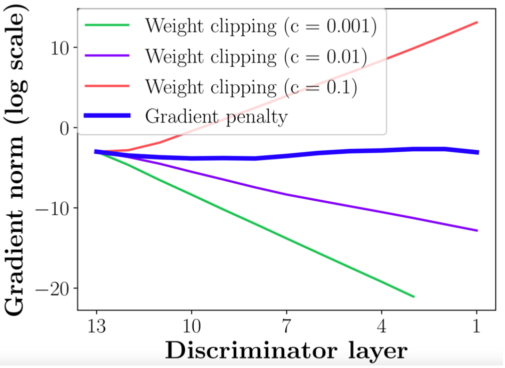
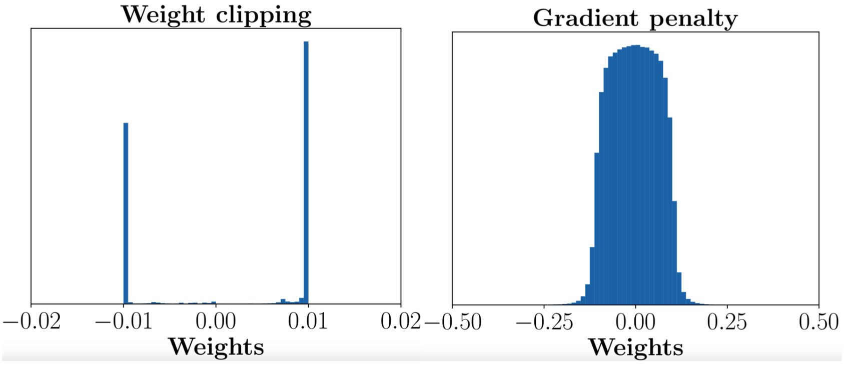


Figure 10: x axis indicate layer's index . Figure 1(b) in Gulrajani et al [17]

In the second problem, weight are bounded on a interval $[-c, c]$, i.e., for those weights greater than c will be set to c, for less than $-c$ will set back to $-c$. This would cause discriminator to learn a simple function. Gulrajani et al verified the problem on Swiss Roll dataset and found that most of weights are distributed either c or $-c$. Instead, the gradient penalty term dose not suffer from this issue, Figure 11 demonstrate this.



(a)

(b)

Figure 11: Figure 1(b) in Gulrajani et al [17]

The above two problem indicate that for the weight clipping method, model performance is too sensitive with respect to the corresponding hypermarameter c.

A differentiable function is K-Lipschitz if and only if the Euclidean norm of it's gradient is almost K everywhere. Instead of using weight clip, Gulrajani et al proposed to add a gradient penalty term which mentioned above to enforce 1-Lipschitz constraint.

$$\left(\left\|\nabla_{\boldsymbol{x}} D(\boldsymbol{x})\right\| - 1\right)^2$$

when discriminator sufficiently trained, the norm of it's gradient will close to 1. Then the new loss function to be minimize becomes

$$L = \underbrace{\mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_g}\big[D(g(\boldsymbol{z}))\big] - \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}\big[D(\boldsymbol{x})\big]}_{\text{original WGAN's loss}} + \underbrace{\lambda\cdot\mathbb{E}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{\boldsymbol{x}}}}\Big[\big(\|\nabla_{\hat{\boldsymbol{x}}}D(\hat{\boldsymbol{x}})\|_2-1\big)^2\Big]}_{\text{new gradient penalty term}}.$$

Where $\hat{\boldsymbol{x}} = \epsilon\boldsymbol{x} + (1-\epsilon)g(\boldsymbol{z})$ and ϵ is a random real number from [0,1]. Thus model can be penalized by the gradient penalty term penalizes if the norm of gradient away from 1.

For experiment setting, Gulrajani et al proposed to not use batch normalization, but layer normalization is recommend. Below is the final algorithm:

Algorithm 4: [Algorithm 1 in Gulrajani et al [17]]. The default hyperparameters are $\lambda = 10, n = 5, \beta_1 = 0, \beta_2 = 0.9, \alpha = 0.0001$, where β_1, β_2 and α are Adam optimizer hyperparameters

```
while before $\theta$ converged do
    for training discriminator $n$ iterations do
        for $i = 1, \cdots,$ batch size $m$ do
            Sample $\boldsymbol{x}$ from data distribution $p_r(\boldsymbol{x})$
            Sample $\boldsymbol{z}$ from latent space $p_g(\boldsymbol{z})$
            Sample random number $\epsilon$ from $U[0,1]$
            $\hat{\boldsymbol{x}} \leftarrow \epsilon\boldsymbol{x} + (1-\epsilon)G_\theta(\boldsymbol{x})$
            $L^{(i)} \leftarrow D\big(G(\boldsymbol{z})\big) - D(\boldsymbol{z}) + \lambda\big(\|\nabla_{\hat{\boldsymbol{x}}}D(\hat{\boldsymbol{x}})\|_2 - 1\big)^2$
        end for
        $w_D \leftarrow w_D - \eta\cdot$ Adam$\Big(\frac{1}{m}\sum_{i=1}^m \nabla_{w_D} L^{(i)}, \beta_1, \beta_2, \alpha\Big)$
    end for
    Sample $k$ minibatch random noise from latent space distribution $p_g(\boldsymbol{z})$
    $w_G \leftarrow w_G - \eta\cdot$ Adam$\Big(-\frac{1}{m}\sum_{i=1}^m \nabla_{w_G} D\big(G(\boldsymbol{z^{(i)}}), \beta_1, \beta_2, \alpha\big)\Big)$
end while
```

4.2 Wasserstein GAN with A Consistency Term (CT-GAN)

Despite the WGAN with gradient penalty term overcome the two problems which exist in the original WGAN, Wei et al. (2017) indicate that the gradient penalty term only works on those interpolated samples $\hat{\boldsymbol{x}}$. At the early training stage, the distribution $\mathbb{P}_g$ is far away from $\mathbb{P}_r$. Moreover, the gradient penalty term usually fail to exam the continuity near real sample $\boldsymbol{x}$. Therefore discriminator can easily out of 1-Lipschitz continuity.

Based on these issues, they introduced to improve the WGAN with gradient penalty by enforce the Lipschitz continuity over the real data $\boldsymbol{x} \sim \mathbb{P}_r$. Specifically, they perturb each real sample $\boldsymbol{x}$ as $\boldsymbol{x}'$ and $\boldsymbol{x}''$, then apply Lipschitz constraint to bound the $D(\boldsymbol{x}')$ and $D(\boldsymbol{x}'')$.

Recall from the WGAN with gradient penalty, a discriminator $D : \mathcal{X} \mapsto \mathcal{Y}$

is Liptschitz continuous if and only if there exists a real number K such that $\forall \boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{X}$,

$$\|D(\boldsymbol{x}_1) - D(\boldsymbol{x}_2)\|_2 \leq K \cdot \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|_2$$

therefore, the following soft consistency term (CT) can be added in order to penalize those violations from the above inequality.

$$CT\Big|_{\boldsymbol{x}_1,\boldsymbol{x}_2} = \mathbb{E}_{\boldsymbol{x}_1,\boldsymbol{x}_2}\left[\max\left(\frac{\|D(\boldsymbol{x}_1) - D(\boldsymbol{x}_2)\|_2}{\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|_2} - M', 0\right)\right]$$

However, Wei et al point out that $\|\boldsymbol{x}_1 - \boldsymbol{x}_2\|$ is impractical, as it is hard to compute all the combinations of $(\boldsymbol{x}_1, \boldsymbol{x}_2)$. Therefore, to make the inequality can be implement, all $(\boldsymbol{x}_1, \boldsymbol{x}_2)$ assumed be bounded by a constant M', and M' is a hyper-parameter. In addition, they proposed to apply dropout before the output of discriminator, denoted as $D(\boldsymbol{x}')$, where $\boldsymbol{x}'$ is sampled from input data, and $D(\boldsymbol{x}'')$ is the second data point which applied in the same corresponding way. $D\_(\cdot)$ denote the output of the second last layer of the discriminator. Thus the final consistency term and loss function is

$$CT\Big|_{\boldsymbol{x}',\boldsymbol{x}''} = \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}\Big[\max\big(0, \|D(\boldsymbol{x}_1) - D(\boldsymbol{x}_2)\|_2\big) + 0.1 \cdot \|(D\_(\boldsymbol{x}', D\_(\boldsymbol{x}'')\|_2 - M'\Big],$$

$$L = \mathbb{E}_{\boldsymbol{z}\sim\mathbb{P}_g}\big[D(G(\boldsymbol{z}))\big] - \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}\big[D(\boldsymbol{x})\big] + \lambda_1 GP\Big|_{\hat{\boldsymbol{x}}} + \lambda_2 CT\Big|_{\boldsymbol{x}',\boldsymbol{x}''}.$$

Final algorithm for the CT-GAN

Algorithm 5: [Algorithm 1 in Wei et al [14]]. The default hyperparameters are $\lambda_1 = 10, \lambda_2 = 2, n = 5, \beta_1 = 0.5, \beta_2 = 0.9, \alpha = 0.0002, m = 64$ where β_1, β_2 and α are Adam optimizer hyperparameters.

```
while before $\theta$ converged do
    for training discriminator $n$ iterations do
        for $i$ = 1, $\cdots$, batch size $m$ do
            Sample $\boldsymbol{x}$ from data distribution $p_r(\boldsymbol{x})$
            Sample $\boldsymbol{z}$ from latent space $p_g(\boldsymbol{z})$
            Sample random number $\epsilon$ from $U[0,1]$
            $\hat{\boldsymbol{x}} \leftarrow \epsilon\boldsymbol{x} + (1-\epsilon)G_\theta(\boldsymbol{x})$
            $L^{(i)} \leftarrow D\big(G(\boldsymbol{z})\big) - D(\boldsymbol{z}) + \lambda_1 GP\big|_{\hat{\boldsymbol{x}}} + \lambda_2 CT\big|_{\boldsymbol{x}',\boldsymbol{x}''}$
        end for
        $w_D \leftarrow w_D - \eta\cdot$ Adam $\left(\frac{1}{m}\sum_{i=1}^{m} \nabla_{w_D} L^{(i)}, \beta_1, \beta_2, \alpha\right)$
    end for
    Sample $k$ minibatch random noise from latent space distribution $p_g(\boldsymbol{z})$
    $w_G \leftarrow w_G - \eta\cdot$ Adam$\Big(-\frac{1}{m}\sum_{i=1}^{m} \nabla_{w_G} D\big(G(\boldsymbol{z}^{(i)}), \beta_1, \beta_2, \alpha\Big)$
end while
```

Chapter 5

Experiments

Describe the experiment in this chapter

5.1 data description

Facebook's AI Research department published pre-trained word vectors for 294 languages in GitHub. They trained them on Wikipedia as source text using the library called fastText. More specifically, the word vectors that were trained using the skip-gram models (Bojanowski et al. 2016) with 300 as embedding dimension. All dataset can be download from their GitHub repository [1].

Conneau et al. showed results trained by Vanilla GAN for some language pairs such as English(en) translate to Spanish(es), English to French(fr) and English to Esperanto(eo) etc. Their code, dictionaries and word embeddings are open source on GitHub (MUSE)[2].

Since the GPU computational resource is limited, and for the convenient of comparing the result and benchmark from the MUSE, we only focus on several more representative language pairs as Conneau et al. (2017) implemented, and plus few other language pairs.

5.2 Multilingual Unsupervised and Supervised Embeddings

In the unsupervised method of MUSE, the linear transformation W across source language embeddings and target embeddings can be learned by GANs without any given dictionary. The experimental setup will be review below.

5.2.1 Network architecture

The vanilla GAN's neural architecture Conneau et al. (2017) proposed is to define the generator as a linear transformation W which maps source embedding matrix X to target embedding matrix Y, therefore the generator is a

[1] FastText: Pre-trained word vectors

[2] MUSE: Multilingual Unsupervised and Supervised Embeddings

single-layer neural network without activation function, the number of nodes in that layer depends on the embedding dimension of language pairs. Since the dimensionality of all monolingual embeddings is 300, hence the generator is a matrix with size 300 × 300, it is a square matrix. The discriminator is a multilayer perceptron with two hidden layers, both have 2048 nodes with Leaky-ReLu as activation function. The output of discriminator is single node squeezed to probability by sigmoid.

5.2.2 Objective functions

Assume $\mathcal{X} = \{\boldsymbol{x}^{(1)}, \cdots, \boldsymbol{x}^{(n)}\}$ and $\mathcal{Y} = \{\boldsymbol{y}^{(1)}, \cdots, \boldsymbol{y}^{(m)}\}$ are source and target language embeddings respectively, the label for source embedding is 1, 0 for target. Since the goal of the discriminator is to identify whether the input coming from source or target. Therefore the discriminator's loss function is

$$\mathcal{L}_D = -\frac{1}{n}\sum_{i=1}^{n} \log P_{\theta_D}\Big(\text{source} = 1 \Big| W\boldsymbol{x}^{(i)}\Big) - \frac{1}{m}\sum_{i=1}^{n} \log P_{\theta_D}\Big(\text{target} = 0 \Big| \boldsymbol{y}^{(i)}\Big)$$

where θ_D denote the parameters of discriminator's network, $P_{\theta_D}\Big(\text{source} = 1 \Big| W\boldsymbol{x}\Big)$ means probability of input point coming from source embeddings, whereas $P_{\theta_D}\Big(\text{target} = 0 \Big| \boldsymbol{y}\Big)$ means target embeddings, i.e., real dataset.

The task of generator is to deceive the discriminator as much as possible, thus, it's loss function is

$$\mathcal{L}_W = -\frac{1}{n}\sum_{i=1}^{n} \log P_{\theta_D}\Big(\text{source} = 0 \Big| W\boldsymbol{x}^{(i)}\Big) - \frac{1}{m}\sum_{i=1}^{n} \log P_{\theta_D}\Big(\text{target} = 1 \Big| \boldsymbol{y}^{(i)}\Big)$$

5.2.3 Optimizer

Since they use minibatch for each training iteration and the batch size is 32, stochastic gradient descent is applied for minimizing $\mathcal{L}_D$ and $\mathcal{L}_W$, the default learning rate is 0.1 without momentum.

5.2.4 Orthogonality

Since the generator is a 300 × 300 matrix, Conneau et al. (2017) proposed to applied an orthogonal constraint to generator after each minibatch iteration. It keep embeddings' quality after source embedding be translated. Moreover, orthogonal matrix have a nice property, assume $\mathcal{O}$ is an orthogonal matrix, i.e. $\mathcal{O}^T\mathcal{O} = \mathcal{O}\mathcal{O}^T = I$, it preserve ℓ_2 norm,

$$\|\mathcal{O}\boldsymbol{x}\|_2^2 = \langle \mathcal{O}\boldsymbol{x}, \mathcal{O}\boldsymbol{x}\rangle = \mathcal{O}^2\langle \boldsymbol{x}, \boldsymbol{x}\rangle = \|\boldsymbol{x}\|_2^2$$

This implies $\|\mathcal{O}\boldsymbol{x}\|_2 = \|\boldsymbol{x}\|_2$, in addition this means $\mathcal{O}$ is an isometry in $\ell 2$ space because it preserve distance.

The algorithm they used to update transformation W is

$$W \longleftarrow (1+\beta)W - \beta(WW^T)W$$

where β is a hyperparameter, as they suggest, $\beta = 0.01$ performs well.

5.2.5 Cross-domain similarity local scaling (CSLS)

A reliable metric is crucial for the criterion of model selection, as we need to compare the similarity between source embeddings and target embeddings.

Nearest neighbors have the asymmetric property, which means that if x is the k-nn of a point y, does not imply that y is a k-nn of x. By Radovanović et al. (2010), the asymmetric property would cause harmful result in high dimensional spaces: For some points, which are called hubs, are nearest neighbors of numerous of other points, however, for some other points, which are called anti-hubs, are even no any points are nearest neighbors of them.

Therefore, Conneau et al. (2017) proposed a bipartite neighborhood graph called Cross-domain similarity local scaling(CSLS) as the new metric. By their definition, the mean cosine similarity from a translated source embedding Wx_s to its k-target neighborhood in target space is defined as

$$r_T(Wx_s) = \frac{1}{k} \sum_{y_t \in \mathcal{N}_T(Wx_s)} \cos(Wx_s, y_t)$$

where $\mathcal{N}_T(Wx_s)$ denote the neighborhood associated with a translated source word Wx_s, and all k words of $\mathcal{N}_T(Wx_s)$ are from target embeddings. Similarly, $\mathcal{N}_S(y_t)$ denote the neighborhood in target embedding corresponded to a target word y_t. The mean cosine similarity of a target word to its k-target neighborhood in translated source embeddings space is defined as

$$r_S(y_t) = \frac{1}{k} \sum_{Wx_s \in \mathcal{N}_S(y_t)} \cos(y_t, Wx_s)$$

these two mean cosine scores for translated source embeddings and target embeddings are implemented by the library called Faiss[3], which greatly speed up the computation of nearest neighbors. The Faiss is developed by Johnson et al. (2017) who are members of Facebook AI research group. The CSLS algorithm define as

$$\text{CSLS}(Wx_s, y_t) = 2\cos(Wx_s, y_t) - r_T(Wx_s) - r_S(y_t).$$

Obviously, there is no any hyperparameter tuning requirement for it. By Conneau et al. (2017) interpretation, the update of CSLS algorithm increase the similarity for those anti-hub word embeddings. In contrast, it also reduce those embeddings in dense hub regions. In addition, the CSLS greatly improves the translation accuracy in experiments.

5.2.6 Evaluation procedure

Conneau et al provides 110 bilingual evaluation dictionaries which are translated by an internal translation tool, each dictionary translate 1500 unique

[3] Faiss: a package for efficient similarity computation.

source words, most of them are commonly used, also polysemy for some vocabularies. After each epoch, the MUSE will calculate the CSLS accuracy on bilingual evaluation dictionary, and report the precision at 1, 5 and 10, i.e., the number of nearest neighbors.

MUSE will also compute the similarity for the top 10,000 frequent words. After finished the calculation of CSLS precision, the length of both source and target embeddings will be normalized to 1, then a subset of source word index which correspond to the temporary dictionary will be selected (source and target word embeddings would be indexed as temporary dictionary before adversarial training), and transformed by learned generator, to match the target word by k-nearest neighbor and CSLS algorithm. At the end of the epoch, the best model will be save based on the mean cosine csls for the maximum of 10000 top frequent embeddings.

5.2.7 dictionary build

1: calculate the knn = 2 (select only one translated target word), or csls 10 for the 10000 most frequent source words in order to get translated words in target space. 2 : For knn, translated saved as a (10000, 2) tensor (all_pairs), the first column is index for the source word, , second column is the closest neighbor. 3: reorder by the different of knn score between 1st closest and 2ed closest, saved as a (10000,2) tensor 4: delete those rows which the target index greater than 10000, then calculated the csls or p@ score

5.2.8 Procruster refinement

Use those most frequent words as anchor points, then apply Procruster algorithm, i.e., compute the SVD(YX^T), where X and Y is source and target embeddings respectively, then use U and V to solve

$$W^* = \operatorname*{argmin}_W \|WX - Y\|_F = UV^T,$$

5.3 Experimental set up

In our experiment, we focused on three improved GANs, i.e., Wasserstain GAN, WGAN with gradient penalty and CT-GAN, to compare the result from vanilla GAN. Our code is based on MUSE, in which use Pytorch as deep learning library instead of TensorFlow. We later found that for WGAN, add layer normalization for the output of fully connected layer can not only increase the accuracy, but also stabilize the training process. Therefore, we would discuss with layer normalization and without layer normalization for these three improved GANs later.

5.3.1 Hyper-parameter search

Since the hyper-parameter search costs lot of GPU resource, therefore, we decide to only to tune those main hyper-parameters which affect most for our

model, for those secondary hyper-parameters such as β_1, β_2 and α in Adam optimizer, we applied the default from PyTorch. We use grid search to find relatively good hyper-parameters for English to Spanish for all three different Algorithms, then extend to another languages pairs.
For Wasserstain GAN, we found $c = 0.25$, and RMSprop with learning rate 0.0005 works well.

For WGAN with gradient penalty, we use $\lambda = 0.5$, and Adam with learning rate 0.0005.

For CT-GAN, we use $\lambda_{GP} = 0.5, \lambda_{CT} = 2$ and $m = 0.2$.

5.3.2 Language pari selection

Søgaard et al. (2018)

5.3.3 Define metric

We use two types of metrics which mentioned at the previous section to select models and hyper-parameters.

- The CSLS nearest neighbor precision at 1, 5 and 10, which evaluate the performance on the evaluation dictionary.
- The mean cosine CSLS, which measure the similarity of the top frequent 10,000 words.

In table 1, the source language is English, we can see that the mean cosine CSLS inaccurately reflect translation performance, e.g. for English to Chinese(zh) and Estonian(et), the precision 10 is very low, but has high overall mean CSLS, this indicate that the translated embeddings have very close neighbors, but almost all of them are incorrect translations. Therefore, we use CSLS nearest neighbor to evaluate translation performance.

Vanilla	es	et	el	fi	hu	tr	pl	zh
p@1	0.398	0.032	0.139	0.103	0.191	0.164	0.203	0.0
p@5	0.678	0.098	0.284	0.263	0.386	0.318	0.465	0.0
p@10	0.750	0.142	0.364	0.358	0.481	0.399	0.574	0.0
mean csls	0.671	0.496	0.560	0.561	0.568	0.555	0.586	0.796

Table 1: p@5, p@10 and mean cosine CSLS for differen target languages

5.3.4 Network architecture

We keep to used the same network architecture for all three improved GANs, only the loss function is different from Vanilla GAN, plus added layer normalization for each layers.

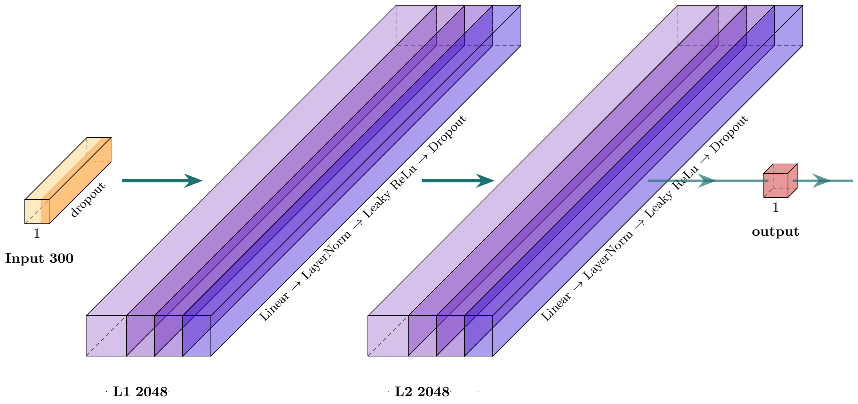

dropout
1
Input 300
Linear → LayerNorm → Leaky ReLu → Dropout
L1 2048
Linear → LayerNorm → Leaky ReLu → Dropout
L2 2048
1
output

Figure 5.1: Architecture of discriminator

5.4 Results

5.4.1 Removed shrinking learning rate with layer normalization

WGAN-GP	en	es	et	el	fi	hu	tr	pl	zh
p@1	83.7	36.3	11.2	13.2	14.5	23	16.2	19.6	5.7
p@5	89.3	64.3	24.2	29.5	34.3	44	31.1	43.8	12.6
p@10	90.8	71.3	30.9	36.6	42.3	52.5	38.5	53.8	16

Table 4: p@1 and p@5, p@10 from different target languages

WGAN	es	et	el	fi	hu	tr	pl	zh
p@1	0.404	0.010	0.077	0.042	0.165	0.0	0.0	0.000
p@5	0.686	0.035	0.193	0.124	0.348	0.000	0.0	0.001
p@10	0.756	0.062	0.265	0.170	0.436	0.001	0.001	0.001

Table 4: p@1 and p@5, p@10 from different target languages

CT-GAN	es	et	el	fi	hu	tr	pl	zh
p@1	0.272	0.060	0.08	0.111	0.165	0.103	0.138	0.002
p@5	0.508	0.149	0.209	0.257	0.340	0.242	0.346	0.011
p@10	0.585	0.196	0.280	0.329	0.419	0.305	0.438	0.018

Table 4: p@1 and p@5, p@10 from different target languages

5.4.2 stability test

Vanilla	**es**	**et**	**el**	**fi**	**hu**	**tr**	**pl**	**zh**
max p@1	39.6	5.7	13.7	13.5	25.6	17.2	24.9	0.0
min p@1	37.9	0.0	12.0	9.1	21.5	10.9	17.3	0.0
avg p@1	39.2	0.6	12.2	11.1	22.7	15.0	19.5	0.0
std p@1	0.7	1.7	1.0	1.6	1.2	2.1	2.6	0.0
max p@5	67.7	15.6	30.0	31.7	45.2	34.0	50.7	0.1
min p@5	63.8	0	25.1	23.4	39.6	24.5	40.1	0.0
avg p@5	66.8	1.6	27.4	27.5	42.4	30.1	43.4	0.0
std p@5	1.2	4.7	1.8	2.5	1.6	3.1	3.4	0.0
max p@10	74.9	22.4	37.7	65.7	54.5	0.103	61.0	0.1
min p@10	71.4	0	31.8	31.3	47.5	30.9	50.3	0.0
avg p@10	74.1	2.3	34.8	35.5	51.3	37.5	53.8	0.1
std p@10	1.1	6.7	2.0	2.9	1.8	3.7	3.5	0.0

Table 4: result of ten runs Vanilla GAN with differen random seed

WGAN-GP	**es**	**et**	**el**	**fi**	**hu**	**tr**	**pl**	**zh**
max p@1	39.6	5.7	13.7	13.5	25.6	17.2	24.9	0.0
min p@1	37.9	0.0	12.0	9.1	21.5	10.9	17.3	0.0
avg p@1	39.2	0.6	12.2	11.1	22.7	15.0	19.5	0.0
std p@1	0.7	1.7	1.0	1.6	1.2	2.1	2.6	0.0
max p@5	67.7	15.6	30.0	31.7	45.2	34.0	50.7	0.1
min p@5	63.8	0	25.1	23.4	39.6	24.5	40.1	0.0
avg p@5	66.8	1.6	27.4	27.5	42.4	30.1	43.4	0.0
std p@5	1.2	4.7	1.8	2.5	1.6	3.1	3.4	0.0
max p@10	74.9	22.4	37.7	65.7	54.5	0.103	61.0	0.1
min p@10	71.4	0	31.8	31.3	47.5	30.9	50.3	0.0
avg p@10	74.1	2.3	34.8	35.5	51.3	37.5	53.8	0.1
std p@10	1.1	6.7	2.0	2.9	1.8	3.7	3.5	0.0

Table 4: result of ten runs Vanilla GAN with differen random seed

5.4.3 without shrinking learning rate

5.4.4 with layer normalization

5.4.5 Wasserstein GAN

5.4.6 result of other people did

5.4.7 comparison tables

5.4.8 visualize network

5.4.9 drawback of the three new WGANs

maybe there is no linear transformation between those languages, because EN to EN ,future work maybe remove the assumption, change the generator to non-linearity

5.4.10 Final experiment

- translate some most frequent words,
- translate some rare words.
- compare how model performs on frequent words, and rare words.

Chapter 6

Conclusions

Appendix A

Some deep learning techniques

A.1 Layer normalization

Training a deep neural network can cause internal covariate shift, i.e., the change of the input distribution of all layers, especially for deep neural network, even a small affects can be expanded when network goes deeper, therefore vanishing gradients can easily happen for saturating non-linearities. Batch normalization can help to reduce internal covariate shift by normalize mini-batch input across corresponding size. In particular,

$$\mu_j = \frac{1}{m}\sum_{j=1}^{n} \boldsymbol{x}_{ij}$$

$$\sigma_j^2 = \frac{1}{m}\sum_{j=1}^{m} (\boldsymbol{x}_{ij} - \mu_i)^2$$

$$\hat{\boldsymbol{x}} = \frac{\boldsymbol{x}_{ij} - \mu_i}{\sqrt{\sigma^2 + \epsilon}},$$

where i indicate the index of example, and j is the index of feature. Thus makes network converge relatively faster, and it is proposed by Loffe et al. (2015).

Despite it achieve great success, batch normalization still have two main drawbacks

- Model performance depends on the mini-batch size, because the mean and variance which calculated by mini-batch cause errors, and these errors will vary from by different mini-batches
- It's very difficult to apply to RNN.

Layer normalization is proposed by Ba et al. (2016). Unlike batch normalization, layer normalization does not depends on the size of mini-batch, and also works well for RNNs. For each input example, it normalize across features in a layer, more specifically,

$$\mu_i = \frac{1}{m}\sum_{j=1}^{m} \boldsymbol{x}_{ij}$$
$$\sigma_i^2 = \frac{1}{m}\sum_{j=1}^{m} (\boldsymbol{x}_{ij} - \mu_i)^2$$
$$\hat{\boldsymbol{x}} = \frac{\boldsymbol{x}_{ij} - \mu_i}{\sqrt{\sigma^2 + \epsilon}}$$

A.2 RMS prop

RMSprop is a optimization algorithm of gradient descent. It is proposed in the lecture 6 of Hinton et al [5] online course, and it is unpublisheded.

The motivation for developing RMSprop is to solve Adagrad's vanishing learning date problem. Unlike other gradient descent algorithm such as sgd or mini-batch gradient descent, by Ruder's explanation, Adagrad can adapts learning rate during training process, but the denominator term tends to zero after some training iterations. In particular, the RMSprop is

$$\mathbb{E}\left[\boldsymbol{g}^2\right]_t = 0.9\mathbb{E}\left[\boldsymbol{g}^2\right]_{t-1} + 0.1\boldsymbol{g}_t^2$$
$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{\eta}{\sqrt{\mathbb{E}\left[\boldsymbol{g}^2\right]_t + \epsilon}}\boldsymbol{g}_t$$

where the ϵ is a small positive number to prevent division equals to zero. The learning rate divided by the mean of gradients which is exponentially decreasing.

A.3 Adam

Adam (Adaptive Moment Estimation) is another popular adaptive gradient descent algorithm, it proposed by Kingma et al. (2015). it combined both RMSprop and momentum, i.e. it not only compute and save the exponentially decaying of previous mean squared gradient, but also compute the previous momentum. More specifically,

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\boldsymbol{g}_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\boldsymbol{g}_t^2$$

[5] Neural Networks for Machine Learning

When m_t and v_t are initialized to zero vectors, Kingma et al. found that the they are unbiased. The final bias corrected terms are

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

combined both

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \frac{\eta}{\sqrt{\hat{v}} + \epsilon}\hat{m}_t$$

Where the default settings are $\beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$. In practice, Adam performs better than other adaptive gradient descent algorithm.

A.4 leaky Relu

The leaky Relu is an non-linear activation, which is developed by Maas et al. (2013). Since the normal Relu is $f(x) = \max(0, x)$, when $f(x)$ is negative, it's gradient will be zero. Therefore, if the output of $f(x)$ mostly are zeros, the normal Relu can cause vanishing gradients. The leaky Relu is to solve this problem by multiply a positive number which is greater than 1, more specifically,

$$f(x) = \begin{cases} x & x \geq 0 \\ \frac{x}{a} & 0 < x \end{cases}$$

where $a \in (1, +\infty)$, thus, the leaky Relu slightly change the gradient of the negative part of $f(x)$'s output, and make the backpropagation not suffered by vanishing gradients.

Bibliography

[1] Sebastian Ruder, Ivan Vulić, and Anders Søgaard. A Survey Of Cross-lingual Word Embedding Models. 2017.

[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. 2013.

[3] Ian J.Goodfellow, Jean Pouget-Abadue, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. 2014.

[4] Yanghua Jin, Jiakai Zhang, Minjun Li, Yingtao Tian, Huachun Zhu, and Zhihao Fang. Towards the Automatic Anime Characters Creation with Generative Adversarial Networks. 2017.

[5] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. 2016.

[6] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralbai. Generating Videos with Scene Dynamics. 2016.

[7] William Fedus, Ian Goodfellow, and Andrew M. Dai. MaskGAN: Better Text Generation via Filling in the______. 2018.

[8] Olof Mogren. C-RNN-GAN: Continuous recurrent neural networks with adversarial training. 2016.

[9] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative Adversarial Text to Image Synthesis. 2016.

[10] Alexis Conneau, Guillaume Lample, Marc'Aurelio Ranzato, Ludovic Denoyer, and Hervé Jégou. Word Translation Without Parallel Data. 2018.

[11] Chao Xing, Dong Wang, Chao Liu, and Yiye Lin. Normalized Word Embedding and Orthogonal Transform for Bilingual Word Translation. 2015.

[12] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN. 2017.

[13] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved Training of Wasserstein GANs. 2017.

[14] Xiang Wei, Boqing Gong, Zixia Liu, Wei Lu, Liqiang Wang. Improving the Improved Training of Wasserstein GANs: A Consistency Term and Its Dual Effect. 2017.

[15] Tomas Mikolov, Quoc V. Le, and Ilya Sutskever. Exploiting Similarities among Languages for Machine Translation. 2017.

[16] Cédric Villani. Optimal transport, old and new. 2008.

[17] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin and Aaron Courville. Improved Training of Wasserstein GANs. 2017.

[18] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching Word Vectors with Subword Information. 2016.

[19] Martin Arjovsky and Léon Bottou. Towards Principled Methods for Training Generative Adversarial Networks. 2017.

[20] Hariharan Narayanan and Sanjoy Mitter. Sample complexity of testing the manifold hypothesis. 2010.

[21] Miloš Radovanović, Alexandros Nanopoulos, and Mirjana Ivanović. Hubs in Space: Popular Nearest Neighbors in High-Dimensional Data. 2010.

[22] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with GPUs. 2017.

[23] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015.

[24] Anders Søgaard, Sebastian Ruder and Ivan Vulić. On the Limitations of Unsupervised Bilingual Dictionary Induction. 2018.

[25] Jimmy Lei Ba, Jamie Ryan Kiros and Geoffrey E. Hinton. Layer Normalization. 2016.

[26] Sebastian Ruder. An overview of gradient descent optimization algorithms. 2017.

[27] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. 2015.

[28] Andrew L. Maas , Awni Y. Hannun and Andrew Y. Ng. Rectifier Nonlinearities Improve Neural Network Acoustic Models. 2013